SQL AND MORE EVENT VALIDATION
Announcements

- Demo Tuesday, April 1
  - Sprint 1
    - Each person chooses a story
    - The team presents the results to the TA
    - The TA grades the results
3

Logging in Users
Tasks to be done

1. Index page with login and new user forms
   1. Create a new session if needed
2. landing page for login (i.e. home page)
   1. retrieve user name and password
   2. check database
      1. add name \$_SESSION[“name”] if valid
      2. return to index.php if not
Tasks to be done

3. landing page for new user

1. add new user to database
How cookies are sent

1. browser requests a Web page

2. server sends page+cookie

Web browser

3. browser requests another page

server
How sessions are established

A diagram illustrating how sessions are established. It shows a flow between a Web Browser, a Web Server, and a Session Store. The process involves:

2. The Web Server responds with a 201 OK status, setting a cookie with PHPSESSID=12345.
3. The Web Browser then sends a GET request to the Web Server for /next.php.
4. The Web Server responding with a 200 OK status, sending HTML content including "Hi William."
functions.php start a session

```php
session_save_path("/var/www/html/nat/login/sessions/");
if (!isset($_SESSION)) { session_start(); }
```
<div id="login">
    <form action="home.php" method="post">
        <legend>Login</legend>
        <input type="text" name="username" value="User Name"/>
        <br/>
        <input type="password" name="password" value="Password"/>
        <br/>
        <input type="submit" value="Login"/>
    </form>
</div>
<?php
    $name = $_REQUEST["name"];
    $email = $_REQUEST["email"];  
    $uname = $_REQUEST["username"];
    $pw = $_REQUEST["password"];
    include("functions.php");
?>

<?php
    if (isset($_REQUEST["name"])) {
        new_user;
    } else {
        log_in($uname, $pw);
    }
?>

function log_in($uname, $password) {
    $db = open_users_database();
    $quoted_uname = $db->quote($uname);
    if (is_password_correct($uname, $password)) {
        $rows = $db->query("SELECT name FROM users WHERE username = "$quoted_uname");
        if ($rows) {
            foreach ($rows as $row) {
                $name = $row["name"];
                $_SESSION["name"] = $name;
            }
        } else {
            redirect("index.php", "Incorrect password or username");
        }
    }
}
function open_users_database() {
    try {
        $db = new PDO("mysql:dbname=login;localhost","nat","<mypassword>"));
    } catch (PDOException $e) {
        print "Error!: " . $e->getMessage() . "<br/>";
        die();
    }
    return $db;
}
function is_password_correct($name, $password) {
    $db = open_users_database();
    $name = $db->quote($name);
    $rows = $db->query("SELECT password FROM users WHERE username = $name");
    if ($rows) {
        foreach ($rows as $row) {
            $correct_password = $row["password"];
            return $password === $correct_password;
        }
    } else {
        return FALSE;  // user not found
    }
}
function redirect($url, $flash_message = NULL) {
    if ($flash_message) {
        $_SESSION["flash"] = $flash_message;
    }
    print $_SESSION["flash"];
    header("Location: $url");
    die();
}
Discuss questions with your Scrum Team
Quiz
1. In PHP, connect to a database with:
   a. `$d = new PDO(...)`
   b. `$d = connect(...)`
   c. `$d = new connection(...)`
   d. `$d = new DB(...)`

2. In PHP, get information with:
   a. `$r = $d->Select(...)`
   b. `$r = $d.Select(...)`
   c. `$r = Select($d, ...)`
   d. `$r = $d->query(...)`

3. In PHP, access the only ‘name’ with:
   a. `$i = $r[0][“name”]`
   b. `$i = $r[0]`
   c. `$i = $r->data[“name”]`
   d. `foreach ($i in r) {$i [“name”]}`

4. In PHP, sanitize db inputs with:
   a. `$s = ‘$input’`
   b. `$s = quote ($input)`
   c. `$s = $d->quote($input)`
   d. `$s = $d->sanitize($input)`
And the answer is ...
Quiz

1. In PHP, connect to a database with: a. $d = new PDO(…)
e.g., $d = new PDO("mysql:dbname=imdb_small", "jessica", "guinness");

2. In PHP, get information with: d. $r = $d->query(…)
e.g., $r = $d->query("SELECT * FROM actors WHERE last_name LIKE 'Del%'");

3. In PHP, access the first ‘name’ with: d. foreach ($i in r) {$i ["name"]
e.g. <?php foreach($rows as $row) {
    echo ‘<li> First name:’ . $row["first_name"] ?> . ‘</li>’;
} ?>

4. In PHP, sanitize db inputs with: b. $s = quote ($input);
e.g., $s = quote($name);
Databases on Betaweb
SQL

- 13.1: Database Basics
- 13.2: SQL
- 13.3: Multi-table Queries
- 13.4: Databases and PHP
- Not in book: Database Design
name = new PDO("dbprogram:dbname=database;host=server", username, password);
$name->query("SQL query");

# connect to world database on local server
$db = new PDO("mysql:dbname=world;host=localhost", "traveler", "packmybags");
$db->query("SELECT * FROM countries WHERE population > 100000000;")

- **PDO** database library lets you connect to many different database programs
  - replaces older, less versatile functions like mysql_connect
- PDO object's query function returns rows that match a query

CSC 210
$db = new PDO("mysql:dbname=world;host=localhost", "traveler", "packmybags");
$rows = $db->query("SELECT * FROM countries WHERE population > 100000000;"STYPE--NED}
foreach ($rows as $row) {
    do something with $row;
}

- **query** returns all result rows
  - As an associative array of [column name -> value]
  - E.g, $row["population"] is row’s population column
A complete example

$db = new PDO("mysql:dbname=imdb_small", "jessica", "guinness");
$rows = $db->query("SELECT * FROM actors WHERE last_name LIKE 'Del%'");
foreach ($rows as $row) {
   .assertNotNull($rows);
    <li> First name: <?= $row["first_name"] ?>, </li>
    Last name:  <?= $row["last_name"] ?> </li>
}
PDO object methods

- query: performs a SQL SELECT query on the database
- exec: performs a SQL query that modifies the database (INSERT, DELETE, UPDATE, etc.)
- getAttribute: gets DB connection properties
- setAttribute: sets DB connection properties
- quote: encodes a value for use within a query
Including variables in a query

Risky: do not directly include variables in a query because they might contain illegal characters or SQL syntax

```php
# get query parameter for name of movie
$title = $_GET["movietitle"];  
$rows = $db->query("SELECT year FROM movies WHERE name = '$title'");
```
Exploits of a Mom (xkcd.com)

HI, THIS IS YOUR SON’S SCHOOL. WE’RE HAVING SOME COMPUTER TROUBLE.

OH, DEAR – DID HE BREAK SOMETHING? IN A WAY –

DID YOU REALLY NAME YOUR SON ROBERT‘? DROP TABLE Students;–?

OH, YES. LITTLE BOBBY TABLES, WE CALL HIM.

WELL, WE’VE LOST THIS YEAR’S STUDENT RECORDS. I HOPE YOU’RE HAPPY.

AND I HOPE YOU’VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.
Quoting variables

# get query parameter for name of movie
$title = $_GET["movietitle"];  
$title = $db->quote($title);  
$rows = $db->query("SELECT year FROM movies WHERE name = $title");

- call PDO's quote method on any variable to be inserted
- quote escapes any illegal chars and surrounds the value with ' quotes
- prevents bugs and security problems in queries containing user input
database commands can often fail (invalid query; server not responding; etc.)

normally, PDO commands fail silently by returning FALSE or NULL

but this makes it hard to notice and handle problems
Exceptions for errors

$db = new PDO("mysql:dbname=imdb_small", "jessica", "guinness");
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
$rows = $db->query("SELECT * FROM movies WHERE year = 2000");  # kaboom!

- using setAttribute, you can tell PDO to throw (generate) a PDOException when an error occurs
- the exceptions will appear as error messages on the page output
- you can catch the exception to gracefully handle the error
Catching an exception

try {
    statement(s);
} catch (ExceptionType $name) {
    code to handle the error;
}

- a try/catch statement attempts to run some code, but if it throws a given kind of exception, the program jumps to the catch block and runs that code to handle the error
Example with error checking

```
try {
    $db = new PDO("mysql:dbname=imdb_small", "jessica", "guinness");
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $rows = $db->query("SELECT * FROM movies WHERE year = 2000");
    foreach ($rows as row) { ... }
} catch (PDOException $ex) {
    <?
    <p>Sorry, a database error occurred. Please try again later.</p>
    <p>(Error details: <?= $ex->getMessage() ?>)</p>
    <?php
}
```
PDOStatement methods

- The $rows returned by PDO's query method is an object of type PDOStatement with these methods:
  - **columnCount()** number of columns in the results
  - **fetch()** return the next row from the results
  - **fetchColumn(number)** return the next column from the results
  - **rowCount()** number of rows returned by the query

```php
if ($db->rowCount() > 0) {
  $first_row = $db->fetch();
  ...
}
```
Database Design

- 13.1: Database Basics
- 13.2: SQL
- 13.3: Multi-table Queries
- 13.4: Databases and PHP
- Not in book: Database Design
Database design principles

- database design: the schema for a database
- database schema: description of table columns and constraints
- some database design principles:
  - keep it simple, stupid (KISS)
  - provide a unique identifier for every row
  - eliminate redundancy
    - redundancy allows inconsistency
  - favor integers for comparisons and repeated values
First database design

<table>
<thead>
<tr>
<th>name</th>
<th>email</th>
<th>course</th>
<th>teacher</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bart</td>
<td><a href="mailto:bart@fox.com">bart@fox.com</a></td>
<td>Computer Science 142</td>
<td>Krabappel</td>
<td>B-</td>
</tr>
<tr>
<td>Bart</td>
<td><a href="mailto:bart@fox.com">bart@fox.com</a></td>
<td>Computer Science 143</td>
<td>Hoover</td>
<td>C</td>
</tr>
<tr>
<td>Milhouse</td>
<td><a href="mailto:milhouse@fox.com">milhouse@fox.com</a></td>
<td>Computer Science 142</td>
<td>Krabappel</td>
<td>B+</td>
</tr>
<tr>
<td>Lisa</td>
<td><a href="mailto:lisa@fox.com">lisa@fox.com</a></td>
<td>Computer Science 143</td>
<td>Hoover</td>
<td>A+</td>
</tr>
<tr>
<td>Lisa</td>
<td><a href="mailto:lisa@fox.com">lisa@fox.com</a></td>
<td>Computer Science 190M</td>
<td>Stepp</td>
<td>A+</td>
</tr>
<tr>
<td>Ralph</td>
<td><a href="mailto:ralph@fox.com">ralph@fox.com</a></td>
<td>Informatics 100</td>
<td>Krabappel</td>
<td>D+</td>
</tr>
</tbody>
</table>

- **good:** simple, can see all data in one place
- **bad:** redundancy (name, email, course repeated)
- **bad:** searches require string comparisons
- **bad:** no unique identifier for each row
## Improved database design

<table>
<thead>
<tr>
<th>students id</th>
<th>name</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Bart</td>
<td><a href="mailto:bart@fox.com">bart@fox.com</a></td>
</tr>
<tr>
<td>456</td>
<td>Milhouse</td>
<td><a href="mailto:milhouse@fox.com">milhouse@fox.com</a></td>
</tr>
<tr>
<td>888</td>
<td>Lisa</td>
<td><a href="mailto:lisa@fox.com">lisa@fox.com</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>teachers id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>Krabappel</td>
</tr>
<tr>
<td>5678</td>
<td>Hoover</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>courses id</th>
<th>name</th>
<th>teacher_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>10001</td>
<td>Computer Science 142</td>
<td>1234</td>
</tr>
<tr>
<td>10002</td>
<td>Computer Science 143</td>
<td>5678</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>grades student id</th>
<th>course_id</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>10001</td>
<td>B-</td>
</tr>
<tr>
<td>123</td>
<td>10002</td>
<td>C</td>
</tr>
<tr>
<td>456</td>
<td>10001</td>
<td>B+</td>
</tr>
<tr>
<td>888</td>
<td>10002</td>
<td>A+</td>
</tr>
</tbody>
</table>
Improvements

- **normalizing**: splitting tables to improve structure / redundancy (linked by unique IDs)
- **primary key**: a column guaranteed to be unique for each record (e.g. Lisa Simpson's ID 888)
- **foreign key**: a column in table A storing a primary key value from table B
  - (e.g. records in grades with student_id of 888 are Lisa's grades)
More Events and Validation
## Page/window events

<table>
<thead>
<tr>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>load</td>
<td>the browser loads the page</td>
</tr>
<tr>
<td>unload</td>
<td>the browser exits the page</td>
</tr>
<tr>
<td>resize</td>
<td>the browser window is resized</td>
</tr>
<tr>
<td>contextmenu</td>
<td>the user right-clicks to pop up a context menu</td>
</tr>
<tr>
<td>error</td>
<td>an error occurs when loading a document or an image</td>
</tr>
</tbody>
</table>
Page/window events

```javascript
// best way to attach event handlers on page load
window.onload = function() { ... };

    document.observe("dom:loaded", function() {
        $("orderform").observe("submit", verify);
    });
```

# Form events

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>submit</td>
<td>Form is being submitted</td>
</tr>
<tr>
<td>reset</td>
<td>Form is being reset</td>
</tr>
<tr>
<td>change</td>
<td>The text or state of a form control has changed</td>
</tr>
</tbody>
</table>

```javascript
window.observe("load", function() {
    \$('orderform').observe("submit", verify);
});

function verify(event) {
    if ($("zipcode").value.length < 5) {
        event.stop(); // cancel form submission unless
    } // zip code is 5 chars long
}  // JS
```
Prototype and forms

\[
\text{\$("id")["name"]}
\]

- gets parameter with given name from form with given id

\[
\text{\$F("id")}
\]

- \$F returns the value of a form control with the given id

```javascript
var name = \$F("username");
if (name.length < 4) {
    \$("username").clear();
    \$("login").disable();
}
```
Prototype and forms

- other form control methods:

  activate   clear   disable   enable
  focus      getValue  present  select
Client-side validation code

```
window.onload = function() {
    $('exampleform').onsubmit = checkData;
};
function checkData(event) {
    if ($('city').value == '' || $('state').value.length != 2) {
        Event.stop(event);
        alert("Error, invalid city/state."); // show error message
    }
}
```

- forms expose onsubmit and onreset events
- to abort a form submission, call Prototype's Event.stop on the event
Regular expressions in JavaScript

- `string.match(regex)`
  - if string fits the pattern, returns the matching text; else returns null
  - can be used as a Boolean truthy/falsey test:
    ```javascript
    var name = $('name').value;
    if (name.match(/[a-z]+/)) { ... }
    ```

- an `i` can be placed after the regex for a case-insensitive match
  - `name.match(/Xenia/i)` will match “xenia”, “XeNiA”, ...
Replacing text with regular expressions

- `string.replace(regex, "text")`
  - replaces the first occurrence of given pattern with the given text
  - `var str = "Xenia Mountrouidou"; str.replace(/[a-z]/, "x")` returns "Xxnia Mountrouidou"

  - returns the modified string as its result; must be stored
    - `str = str.replace(/[a-z]/, "x")`

- A `g` can be placed after the regex for a global match (replace all occurrences)
  - `str.replace(/[a-z]/g, "x")` returns "Xxxxx Mxxxxxxxxxxxxx"
## Keyboard/text events

<table>
<thead>
<tr>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>keydown</td>
<td>user presses a key while this element has keyboard focus</td>
</tr>
<tr>
<td>keyup</td>
<td>user releases a key while this element has keyboard focus</td>
</tr>
<tr>
<td>keypress</td>
<td>user presses and releases a key while this element has keyboard focus</td>
</tr>
<tr>
<td>focus</td>
<td>this element gains keyboard focus</td>
</tr>
<tr>
<td>blur</td>
<td>this element loses keyboard focus</td>
</tr>
<tr>
<td>select</td>
<td>this element's text is selected or deselected</td>
</tr>
</tbody>
</table>
# Key event objects

<table>
<thead>
<tr>
<th>property name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>keyCode</td>
<td>ASCII integer value of key that was pressed (convert to char with <code>String.fromCharCode</code>)*</td>
</tr>
<tr>
<td>altKey, ctrlKey, shiftKey</td>
<td>true if Alt/Ctrl/Shift key is being held</td>
</tr>
</tbody>
</table>

*rototype's key code constants
### Key event objects

<table>
<thead>
<tr>
<th>Event.KEY_BACKSPACE</th>
<th>Event.KEY_DELETE</th>
<th>Event.KEY_DOWN</th>
<th>Event.KEY_END</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event.KEY_ESC</td>
<td>Event.KEY_HOME</td>
<td>Event.KEY_LEFT</td>
<td>Event.KEY_PAGEDOWN</td>
</tr>
<tr>
<td>Event.KEY_PAGEUP</td>
<td>Event.KEY_RETURN</td>
<td>Event.KEY_RIGHT</td>
<td>Event.KEY_TAB</td>
</tr>
<tr>
<td>Event.KEY_UP</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Issue:** if the event you attach your listener to doesn't have the focus, you won't hear the event

- **Possible solution:** attach key listener to entire page body, outer element, etc.